Obstacles: Pretty much a given of needing a for loop to traverse every array, for some arrays needing to use multiple for loops to compare values multiple times.

Realized that somethings I was trying to do in one function was a capability of the other function, so I was able to use rotateLeft() in split().

this string is used through rotateLeft

string names[9] = {"james","edward","sean","frank","mary","liz","theresa","mary","tony"};

appendToAll:

assert(appendToAll(names, 0, " is a person") == 0);

assert(appendToAll(names, 5, " is a person") == 5);

assert(appendToAll(names, -10, " is a person") == -1);

then check all values to see if they have been appended.

lookup:

assert(lookup(names, 9, "james") == 0);

assert(lookup(names, -4, "james") == -1);

assert(lookup(names, 9, "jame") == -1);

assert(lookup(names, 9, "mary") == 4);

assert(lookup(names, 3, "frank") == -1);

assert(lookup(names, 9, "jill") == -1);

positionOfMax:

assert(positionOfMax(names, 9) == 8);

assert(positionOfMax(names, 5) == 2);

assert(positionOfMax(names, 0) == -1);

assert(positionOfMax(names, -3) == -1);

finds greatest string value on first character, based on ASCII.

rotateLeft:

assert(rotateLeft(names, 9, 3) == 3);

assert(rotateLeft(names, 9, 9) == -1);

assert(rotateLeft(names, 9, 8) == 8);//if pos is one less than n then nothing should change

assert(rotateLeft(names, -1, 8) == -1);

assert(rotateLeft(names, 9, 0) == 0);

assert(rotateLeft(names, 9, -1) == -1);

check that list is moving accurately for the ones that succeed.

countRuns:

Array, N, Output

{"james","edward","sean","frank","mary","liz","theresa","mary","tony"};

{ "david", "david", "calvin", "jamie", "jilly", "jilly" };

{ "david", "jamie", "calvin", "jamie", "jilly", "jilly" };

{ "david", "david", "jilly", "jamie", "jamie", "jilly" };

|  |  |  |
| --- | --- | --- |
| ARRAY | N VALUE | OUTPUT |
| {"james","edward","sean","frank","mary","liz","theresa","mary","tony"}; | 9  4 | 9  4 |
| { "david", "david", "calvin", "jamie", "jilly", "jilly" }; | 6  -3 | 4  -1 |
| { "david", "jamie", "calvin", "jamie", "jilly", "jilly" }; | 6  0 | 5  0 |
| { "david", "david", "jilly", "jamie", "jamie", "jilly" }; | 3  6 | 2  4 |

flip:

|  |  |  |
| --- | --- | --- |
| ARRAY | N VALUE,  RETURN | NEW ARRAY |
| {"james","edward","sean","frank","mary","liz","theresa","mary","tony"}; | 9,9 | "tony", "mary", "theresa", "liz", "mary", "frank", "sean", "edward", "james"}; |
| { "david", "david", "calvin", "jamie", "jilly", "jilly" }; | 6,6 | "jilly", "jilly", "jamie", "calvin", "david", "david" |
| Same as above | -3,-1 | unchanged |
| Same as above | 5,5 | "jilly", "jamie", "calvin", "david", "david", "jilly" |
| Same as above | 0,0 | unchanged |

differ:

string list1[9] = {"james","edward","sean","frank","mary","liz","theresa","mary","tony"};

string list2[9] = {"james","edward","sean","frank","mary","justin","theresa","mary","tony"};

assert(differ(list1, 9, list2, 9) == 5);

assert(differ(list1, 9, list2, 3) == 3);

assert(differ(list1, 3, list2, 9) == 3);

assert(differ(list1, -1, list2, 9) == -1);

assert(differ(list1, 4, list2, -1) == -1);

assert(differ(list1, 4, list2, 0) == 0);

subsequence:

string list1[9] = {"james","edward","sean","frank","mary","liz","theresa","mary","tony"};

string list2[2] = {"mary","liz"};

string list3[5] = {"edward","sean","frank","mary","liz"};

assert(subsequence(list1, 9, list2, 2) == 4);

assert(subsequence(list1, 9, list2, 1) == 4);

assert(subsequence(list1, 9, list2, 0) == 0);

assert(subsequence(list1, 9, list3, 2) == 1);

assert(subsequence(list1, 9, list3, -1) == -1);

assert(subsequence(list1, -1, list2, 2) == -1);

lookupAny:

string list1[9] = {"james","edward","sean","frank","mary","liz","theresa","mary","tony"};

string list2[6] = {"justin", "calvin", "brayeden","liz", "samantha", "marissa"};

string list3[5] = {"edward","sean","frank","mary","liz"};

assert(lookupAny(list1, 9, list2, 6) == 5);

assert(lookupAny(list1, 9, list2, 2) == -1);

assert(lookupAny(list1, 9, list3, 5) == 1);

assert(lookupAny(list1, 9, list2, 0) == -1);

assert(lookupAny(list1, -1, list2, 6) == -1);

assert(lookupAny(list1, 1, list2, 6) == -1);

split:

assert(split(list1, 9, "abraham") == 0);

assert(split(list1, 9, "greg") == 2);

Edward and frank before

assert(split(list1, 9, "kris") == 3);

now includes james

assert(split(list1, 9, "nigel") == 6);

includes mary liz and mary

assert(split(list1, 9, "xander") == 9);

now includes the rest of the list

assert(split(list1, -1, "xander") == -1);

assert(split(list1, 0, "xander") == 0);